99-339
1496.00059

CONFIGURABLE HARDWARE REGISTER STACK FOR CPU ARCHITECTURES

Field of the Invention

The present invention relates to a processor generally and, more particularly, to a configurable hardware register stack used by the processor.

Background of the Invention

Conventional hardware register stacks associated with a central processing unit (CPU) are implemented so that a predetermined subset of general registers, or all of the general registers within the hardware register stack, are stacked when doing a subroutine call. The subroutine calls are started and ended by jump and branch instructions. The general registers are pushed onto the hardware register stack before executing the subroutine. The general registers are popped off the hardware register stack before leaving the subroutine.

An instruction set architecture (ISA) of the CPU provides specific definitions for some general registers. For example, one general register is a link register. The link register saves an

address of the jump or the branch instruction that caused the subroutine to be executed. Consequently, the ISA-specific general registers should not be used by executable code.

Code compilers use the ISA-nonspecific general registers for predetermined purposes. The code compilers, however, commonly use only a portion of the available general registers. As a result, pushing and popping from the hardware register stack is inefficient. The general registers unused by the ISA and by the code compilers are pushed and popped from the hardware register stack though they do not contain useful information. Furthermore, the unused general registers cannot be used to hold global information that is common to multiple subroutines because the values change each time a push or a pop instruction occurs.

Predeterminations in the ISA and the code compilers for stacking of the general registers result in less-than-optimal performances for stacking operations. Speed of time critical applications such as exceptions handling routines and interrupt handling routines is limited by the stacking operations. Furthermore, application of multiple ISAs by the CPU is often difficult or impossible because of the limitations imposed by the ISA-specific general registers.

Summary of the Invention

The present invention concerns a circuit comprising a register stack and a control circuit. The register stack may be configured as (i) a plurality of segments addressable through a segment address signal and (ii) a plurality of registers within each of the plurality of segments. The plurality of registers are generally addressable through a register address signal. The control circuit may be configured to (i) store a plurality of register states, (ii) store a segment count signal, and (iii) present the segment address signal responsive to the plurality of register states, the segment count signal, and the register address signal.

The objects, features and advantages of the present invention include providing a method and/or a hardware register stack and controller that may (i) be programmed to fit different code compilers using different general registers when doing subroutine calls, (ii) execute fast subroutine calls, and/or (iii) speed up context switching when changing between different tasks while adding very little extra control overhead to the existing ways of implementing a register stack.

Brief Description of the Drawings

These and other objects, features and advantages of the present invention will be apparent from the following detailed description and the appended claims and drawings in which:

FIG. 1 is a block diagram illustrating a central processing unit (CPU) in accordance with a preferred embodiment of the present invention;

FIG. 2 is a more detailed block diagram illustrating an example of the CPU;

FIG. 3 is a flow diagram of a method for controlling a register stack;

FIG. 4 is a detailed block diagram of a preferred embodiment of a status circuit within the CPU;

FIG. 5 is a detailed block diagram of an alternative embodiment of the status circuit;

FIG. 6 is a block diagram of an alternative embodiment of the present invention; and

FIG. 7 is a flow diagram of a method for using the register stack.

Detailed Description of the Preferred Embodiments

Referring to FIG. 1, a block diagram illustrating a central processing unit (CPU) 100 implementing a preferred embodiment of the present invention is shown. The CPU 100 generally comprises a stack control circuit 102, a register stack 104, and a stack register 105. The stack control circuit 102 and the register stack 104 may be in communications with software or code 106 being executed by the CPU 100.

The code 106 may present an address signal (e.g., REG_ADDR) to an input 108 of the stack control circuit 102 and an input 110 of the register stack 104. The stack register 105 may have an output 111 that may present a data signal (e.g., REG_STATES) to an input 112 of the stack control circuit 102. The signal REG_STATES may be set by the code 106 coming out of a reset handler operation for the CPU 100. The stack control circuit 102 may have an output 114 that may present an address signal (e.g., SEG_ADDR) to an input 116 of the register stack 104.

Referring to FIG. 2, a more detailed block diagram illustrating an example of the CPU 100 is shown. The stack control circuit 102 may comprise a status circuit 118, a counter 120, and a number of gates 122. The gates 122 may be implemented as logical

AND gates. However, other types of gates may be implemented to meet the design criteria of a particular application. Generally, the hardware required to implement the stack control circuit 102 may be very small.

The register stack 104 may be implemented as a block of memory mapped registers. The register stack 104 may comprise multiple segments 124A-N. Each segment 124A-N may further comprise multiple registers 126A-R. The multiple registers 126A-R are illustrated only in a third segment 124C of the register stack 104 for clarity. The register stack 104 may be implemented using synchronous and/or asynchronous type random access memory (RAM) devices.

The signal REG_ADDR may be presented to the input 108 of the status circuit 118. The status circuit 118 generally has an output 128 that may present a gating signal (e.g., STACK_GATING) to inputs 130 of the gates 122. The signal REG_STATES is generally received at the input 112 of the status circuit 118. The counter 120 generally has an output 132 for presenting a count signal (e.g., SEG_COUNT) to inputs 134 of the gates 122. The signal SEG_ADDR may be presented at the output 114 by the gates 122.

The signal REG_STATES may have one symbol or one bit associated with each general register 126A-R of the CPU 100. Each symbol or bit of the signal REG_STATES may have one of two states, a global state or a stackable state. The global state generally indicates that the associated general register 126A-R is not to be pushed onto the register stack 104 upon entering a subroutine call, nor popped from the register stack 104 at the end of the subroutine call. The general registers 126A-R having the global state may be available to all subroutines and thus may provide a convenient mechanism to pass data among subroutines. In a preferred embodiment, at least one general register 126A-R may be fixed with the global state. For example, in the MIPS architecture (MIPS Technologies, Inc. of Mountain View, California), a bottom register (R0) is wired to present a value of zero always. Since the bottom register (R0) cannot change values, the bottom register may be assigned the global state and made available to all subroutines without having to be repeated in all segments 124A-N of the register stack 104.

The stackable state generally indicates that the associated general register 126A-R should be pushed and popped to and from the register stack 104. Data stored in a stackable

general register 126A-R is generally changeable only by the calling subroutine. In a preferred embodiment, at least one general register 126A-R may be fixed with the stackable state. For example, the MIPS architecture defines the top register (R31) as a link register. As part of a push instruction, a return address is generally stored in the link register. Using the present invention, the link register may be pushed by incrementing the signal SEG_ADDR prior to writing the return address into the link register. The return address may be popped by reading the return address from the link register prior to decrementing the signal SEG_ADDR. Consequently, the top register (R31) should always be stacked in the MIP architecture since the return address from the subroutine should be saved.

The status circuit 118 may also receive the signal REG_ADDR. The signal STACK_GATING may be presented by the status circuit 118 based upon the state of the symbol or bit stored in the status circuit 118 associated with the general register 126A-R identified by the signal REG_ADDR. A combination of the signal REG_ADDR and the signal SEG_ADDR is generally used to identify one register 126A-R within one segment 124A-N of the register stack 104 being accessed.

Referring to FIG. 3, a flow diagram of a method for controlling the register stack 104 is shown. Operationally, the signal REG_STATES may be set or loaded into the stack register 105 and presented to the status circuit 118 (e.g., block 136). The counter 120 presents the signal SEG_COUNT to identify the current segment 124A-N at the top of the register stack 104 (e.g., block 138). The counter 120 is normally initialized to zero, indicating that the segment zero 124A is the initial top of the register stack 104. Upon execution of a push instruction, the counter 120 generally increments the signal SEG_COUNT. Incrementing the signal SEG_COUNT effectively moves the top of the register stack 104 up one segment 124A-N. If the signal SEG_COUNT is already at the top of the register stack 104 (e.g., segment 124N) when the push instruction is executed, then a normal exception routine may be invoked.

Upon execution of a pop instruction, the counter 120 generally decrements the signal SEG_COUNT. Decrementing the signal SEG_COUNT effectively moves the top of the register stack 104 down one segment 124A-N. If the signal SEG_COUNT is already at the bottom segment (e.g., segment zero 124A) when the pop instruction is executed, then a normal exception routine may be invoked.

The signal STACK_GATING may be compared with the signal SEG_COUNT to present the signal SEG_ADDR (e.g., block 140). In a preferred embodiment, if the signal STACK_GATING is in the global state (e.g., the GLOBAL branch of decision block 142), then the signal SEG_ADDR may be presented as a predetermined segment address (e.g., block 144). In particular, the signal STACK_GATING may use a logical low (e.g., a digital zero) as the global state and a logical high (e.g., a digital one) as the stackable state. Using the above convention, the signal STACK_GATING in the global state causes the signal SEG_ADDR to be presented as all logical zeros, thus addressing the segment zero 124A. Likewise, if the signal STACK_GATING in the stackable state (e.g., the STACKABLE branch of decision block 142), then the signal SEG_ADDR may be set to the signal SEG_COUNT (e.g., block 146). Once the signal SEG_ADDR has been determined, the appropriate general register 126A-N of the appropriate segment 124A-N of the register stack 104 may be accessed (e.g., block 148). A general delay time from receiving the signal REG_ADDR to having the proper register 126A-R accessible may be less than one clock cycle in a preferred embodiment. Other delay times may be implemented to meet the design criteria of a particular application.

Referring to FIG. 4, a detailed block diagram illustrating a preferred embodiment of the status circuit 118 is shown. The status circuit 118 may be implemented as a comparator 152. The comparator 152 may have the input 108 that may receive the signal REG_ADDR and the input 112 that may receive the signal REG_STATES. The comparator 152 may also have the output 128 for presenting the signal STACK_GATING. The comparator is generally configured to select the state of one general register 126A-R from the signal REG_STATES, based upon the signal REG_ADDR. The selected state may be presented as the signal STACK_GATING.

Referring to FIG. 5, a detailed block diagram of an alternative embodiment of the status circuit 118 is shown. A random access memory (RAM) device 158 may be used to store the signal REG_STATES and present the signal STACK_GATING. The RAM 158 may comprise R-by-1 bit addressable memory cells. Each memory cell generally stores the global/stackable state for one general register 126A-R. The RAM 158 may have a one-bit output (e.g., the output 128) addressed using the signal REG_ADDR. The one-bit output generally presents the signal STACK_GATING.

Referring to FIG. 6, a block diagram of an alternative embodiment of the present invention is shown. In this alternative

embodiment, the register stack 104 may be physically implemented in two portions, one portion remotely located from the other portion. A first portion 104A of the register stack 104 may be physically implemented inside the CPU 100 as before. A second portion 104B of the register stack 104 may be physically implemented external to the CPU 100. Consequently, the CPU 100 may have an output 164 and another output 166 to present the signal REG_ADDR and the signal SEG_ADDR respectively to the second portion 104B. The first portion 104A may have an input 162A to receive the signal REG_ADDR and an input 116A to receive the signal SEG_ADDR. Likewise, the second portion 104B may have an input 162B to receive the signal REG_ADDR and an input 116B to receive the signal SEG_ADDR.

Each of the first portion 104A and the second portion 104B are generally implemented as one or more segments 124. An advantage of implementing the second portion 104B external to the CPU 100 is that the second portion 104B may be customized to the design criterial of a particular application. Where the particular application requires a minimal register stack 104, then the external portion 104B may be implemented with only a few or even no segments 124. Were the particular application requires a large register stack 104, then the external portion 104B may be

implemented with many segments 124 without impacting the design of the CPU 100. In one implementation, the internal portion 104A may be eliminated and the entire stack register 104 may be the external portion 104B.

Referring to FIG. 7, a flow diagram of a method for using the stack register 104 is shown. Upon execution of a push instruction for a subroutine call, the signal SEG_ADDR may be incremented (e.g., block 168). The CPU 100 may then execute the code 106 for the subroutine call (e.g., block 170). Upon completion of the subroutine call, the signal SEG_ADDR may be decremented (e.g., block 172).

The ability of the code 106 to change the global/stackable state of individual general registers 126A-R generally allows for customization of the register stack 104. For example, the register stack 104 may be configured to optimize specific subroutine calls intended to execute quickly, such as interrupt and exception handlers. By implementing the register stack 104 internal to the CPU 100, the general register 126A-R may be stacked in one clock cycle or less without requiring any stall cycles. In another example, the ability to program the global/stackable state of the general registers 126A-R generally

allows for code 106 compiled by different compilers to be executed by the same CPU 100. Each time code 106 from a different compiler is to be executed, the global/stackable state of each general register 126A-R may be changed to match the compiler. This may be well suited for operating systems to speed up context switching when changing between different tasks.

The various signals of the present invention are generally "on" (e.g., a digital HIGH, or 1) or "off" (e.g., a digital LOW, or 0). However, the particular polarities of the on (e.g., asserted) and off (e.g., de-asserted) states of the signals may be adjusted (e.g., reversed) accordingly to meet the design criteria of a particular implementation.

The function performed by the flow diagrams of the figures may be implemented as emulations, simulations and/or models using a conventional general purpose digital computer programmed according to the teachings of the present specification, as will be apparent to those skilled in the relevant art(s). Appropriate software coding can readily be prepared by skilled programmers based on the teachings of the present disclosure, as will also be apparent to those skilled in the relevant art(s).

The present invention may also be implemented by the preparation of ASICs, FPGAs, or by interconnecting an appropriate network of conventional component circuits, as is described herein, modifications of which will be readily apparent to those skilled in the art(s).

The present invention thus may also include a computer product which may be a storage medium including instructions which can be used to program a computer to perform a process in accordance with the present invention. The storage medium can include, but is not limited to, any type of disk including floppy disk, optical disk, CD-ROM, and magneto-optical disks, ROMs, RAMs, EPROMs, EEPROMs, Flash memory, magnetic or optical cards, or any type of media

While the invention has been particularly shown and described with reference to the preferred embodiments thereof, it will be understood by those skilled in the art that various changes in form and details may be made without departing from the spirit and scope of the invention.